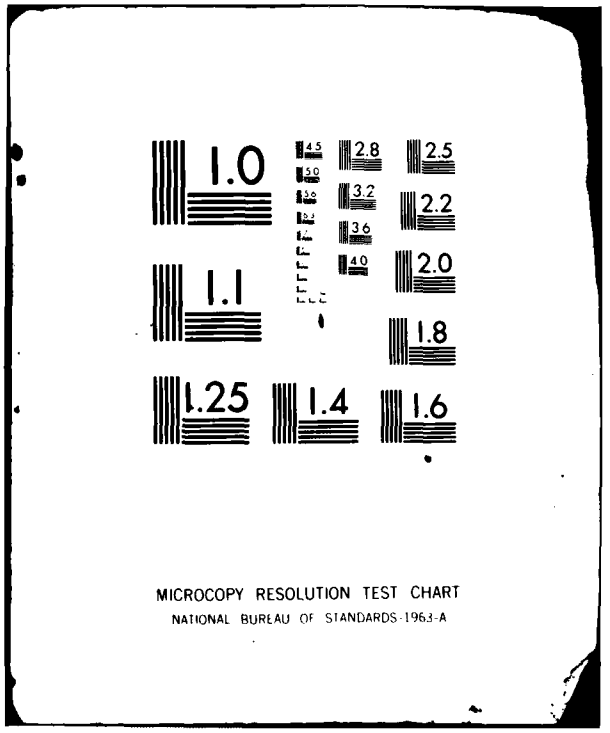
AD-A111 954 STATE UNIV OF NEW YORK AT BUFFALO AMHERST DEPT OF CO--ETC F/G 9/2
A SIMPLE 'TEACHABLE AUTOMATIC PROGRAM-GENERATING SYSTEM' WRITTE--ETC(U)
DEC 81 N V FINDLER AFOSR-81-0220
UNCLASSIFIED AFOSR-TR-82-0093 NL

END
DATE FILMED
2-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Technical (3)

A SIMPLE 'TEACHABLE AUTOMATIC PROGRAM-GENERATING SYSTEM' WRITTEN AS A TERM PROJECT IN AN INTRODUCTORY COURSE ON ARTIFICIAL INTELLIGENCE*

Nicholas V. Findler
Department of Computer Science
State University of New York at Buffalo

Published in the Proceedings of the Third World Conference on Computer Education, Lausanne, Switzerland, pp. 725-729, 1981.

*The writing up of this paper was supported by the AFOSR [redacted] 81-0220. The work described was done mainly while the author was a Visiting Professor at the Free University of Amsterdam and the University of Amsterdam, The Netherlands during his sabbatical year in 1979-80.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0093	2. GOVT ACCESSION NO. AD-A111 754	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A SIMPLE 'TEACHABLE AUTOMATIC PROGRAM-GENERATING SYSTEM' WRITTEN AS A TERM PROJECT IN AN INTRODUCTORY COURSE ON ARTIFICIAL INTELLIGENCE	5. TYPE OF REPORT & PERIOD COVERED TECHNICAL 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Nicholas V. Findler	8. CONTRACT OR GRANT NUMBER(s) AFOSR-81-0220
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science State University of New York at Buffalo 4226 Ridge Lea Road, Amherst NY 14226	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332	12. REPORT DATE DEC 81 13. NUMBER OF PAGES 12
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

PUBLISHED IN THE PROCEEDINGS OF THE THIRD WORLD CONFERENCE ON COMPUTER EDUCATION, LAUSANNE, SWITZERLAND, pp. 725-729, 1981

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A fairly flexible specification for a "Teachable Automatic Program-Generating System' was given as the subject matter of a term project to undergraduate students in an introductory course on Artificial Intelligence. It was pointed out that good design ideas and feasible-looking methods are more important to submit than perfectly-running but dull programs.

The majority of students formed teams of 3-4 members and produced interesting,
(CONTINUED)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONT.:

but very often incomplete, systems within a time period of about 10 weeks. It was found that the students' insecurity due to the lack of useable approaches available in the literature was more than balanced by the pride they took in working on original and creative research at the beginning of their professional career.

ABSTRACT

A fairly flexible specification for a 'Teachable Automatic Program-Generating System' was given as the subject matter of a term project to undergraduate students in an introductory course on Artificial Intelligence. It was pointed out that good design ideas and feasible-looking methods are more important to submit than perfectly-running but dull programs.

The majority of students formed teams of 3-4 members and produced interesting, but very often incomplete, systems within a time period of about 10 weeks. It was found that the students' insecurity due to the lack of useable approaches available in the literature was more than balanced by the pride they took in working on original and creative research at the beginning of their professional career.

INTRODUCTION

Just over 10 years ago, the present author reported on the course contents and teaching techniques used in courses on non-numerical computation [1]. Many significant advances have since been made in the different areas of Computer Science, which are well reflected in the latest course recommendations of the Curriculum Committee of ACM [2].

In this paper, we shall describe the experience relating to a term project given out in an introductory course on Artificial Intelligence. The course was offered to senior undergraduate students of both the Free University of Amsterdam and the University of Amsterdam, The Netherlands while the author was a Visiting Professor at these institutions during his sabbatical year in 1979-80. The students' level of preparation was comparable to that of senior undergraduate/junior graduate students in the United States. Teams of 3-4 students were formed although some of the students preferred to work either with only one other student or by himself/herself.

Because of their good programming background in high-level languages, such as PASCAL, ALGOL 68 and LISP, and a good grasp of the task (to be given below), the students could start working on the term project after the first 3-4 weeks of class. The grade in the course was determined by the weighted sum of the scores the student had received on the term project (identical for each member within a team), the mid-term and the final examinations -- the respective weights being 40, 25 and 35.

At first, it was thought that the policy of giving identical grades to every member of a team may induce good students to join students of only similar standing. However, this turned out not to be

the case -- there were some but not significant differences between the final course grades within teams. It must be noted though that our sample is somewhat biased as only better motivated students took this course, an elective course offered at rather irregular times.

Finally, there was free exchange of ideas between the teams, both during class discussions and outside class. Although the techniques of the teams differed from one another in several ways, a certain adaptation of approaches could at times be noticed in the final products.

THE SPECIFICATION OF THE PROBLEM

The following text was given to the students:

(a) The General Problem Area

The redundant and statistically distributed logic in our brains contributes to its significant surviving capability and to the observed fact that an initial impairment of brain functioning (due to, for example, an automobile accident or war wounds), often and to a large extent, gradually disappears. In contrast, there is relatively little error detection and correction in standard computer hardware and, particularly, software -- often for reasons of ill-conceived economy.

It would be desirable to construct a "sufficiently" reliable programming system. By this, we mean that if a computational process is interrupted (e.g. programs and data become temporarily garbled due to a transient external change in the operating environment), the supervisory system takes control of operations and attempts to reconstruct the garbled information until the impairment is either eliminated or bypassed.

(b) The Teachable Automatic Program-Generating System (TAPGS)

The first step toward such self-repairing system is the creation of a subsystem to which the user can specify tasks either in a formal language or in a subset of English. In the task specification, the user would refer to previously accomplished tasks which utilize program building blocks (PBB's) and procedures built of PBB's so far. The level of complexity of the tasks is increasing as experience is gained. In cases of insufficient knowledge held by the system, it can request and receive hints and guidance from the user.

Let us call KID our primitive TAPGS, visualized as a 7-8 year old child (it can read/write, understand directions, etc.). It will be taught to accomplish some simple tasks in an environment consisting of, for example, persons, places and objects. The training and the issuing of commands are done by TEACH (the user). The language of communication is called TALK (TEACH's Algorithmic Language for KID).

Without going into the details of a possible structure, we note that (using the above example) the State Description of an object x, S(x), consists of its location, L(x), and the name of the object holding it directly, H(x). That is S(x)=(L(x), H(x)). The value of H(x) when x is not held is NULL. There are as many additional attribute-value pairs as you only may wish to specify.

Further, each PBB and procedure built from them must have attached to it: (a) a simple task environment in which its proper functioning can be tested; (b) statistical information concerning how many times it has been invoked and by which other procedure, and how many times it has invoked which other procedure; (c) pre-conditions, post-conditions and goal-conditions (see their definitions below).

A Program Building Block is either a Primitive Program Function (PPF) or a Primitive Program Control (PPC), both directly understandable by KID. The PPC determines the order of execution of and the information flow between PPF's when several are combined as a consequence of the training process. The execution of only one or several PPF's causes a change in the State Description of some objects. Before we give examples of PPF's, we have to discuss three more concepts: Pre-Conditions, Post-Conditions and Goal-Conditions. These will localize errors for easier detection in the second (for us now irrelevant) phase of the project and also make sure that a partial failure will not cause global disaster.

Both Pre-Conditions and Post-Conditions are statements in the body of the definition of a PPF and compare values according to some Criterion Operators. The remaining parts of the PPF are permitted to be executed only when a Pre-Condition succeeds. Similarly, the whole PPF succeeds only when the Post-Condition has also succeeded at the end. The Goal-Condition is an additional sort of Pre-Condition but it tests the suitability of the PPF for a particular goal rather than its invocability in a given environment. We envisage the following Criterion Operators: EQ, NE, LE, LT, GE, GT, MAX, MIN, AT.

The table on the next page provides a tentative list of PPF's. You are welcome to add to it or leave off some of it. The following comments are also needed. 'TEXT' is a data type to be used only for input/output and become quoted automatically. The 'temporary' data type is used for KID's referring to items whose names it does not know yet. All declarations and definitions by TEACH are global. A Pre-Condition such as 'x is pickupable' implies several expressible criteria in this case: not both of KID's hands are full and the

weight of x is below a certain limit.

A high-level inductive generalization should also be considered. (It would, among other things, improve memory utilization.) From the specifications given in TALK, the system would build several Abstract Task Descriptions for similar functions which have somewhat different arguments, Pre-Conditions and Post-Conditions. At regular times, the monitor, acting like a garbage-collector, sweeps through the canonical definitions of procedures and combines the appropriate ones into a higher level, generalized version. An example of this is the merging and generalizing of the already acquired and tested routines BUYSUGAR (place) and BUYMILK (place) into BUY (item, place). In contrast, the user may decide to make a much too general routine more specific and simpler by keeping some of its input values at a frequently used constant level.

Emphasis is placed on working out good design ideas and feasible approaches rather than on perfectly running but dull programs.

Primitive Program Function:	Precondition:	Postcondition:
GOTO(L(x))	L(KID)≠L(x)	L(KID)=L(x)
PUSH(x,L')	L(KID)=L(x)≠L' x is moveable	L(KID)=L(x)=L'
PICKUP(x)	L(KID)=L(x) x is pickupable	H(x)=KID
GIVE(x,y)	H(x)=KID L(KID)=L(y) x is holdable by y	H(x)=y
PUTDOWN(x)	H(x)=KID	H(x)=NULL L(x)=L(KID)
ENTER(x)	L(KID)=L(x) x is enterable	L(KID)=L(ENTRANCE(x))
LEAVE(x)	L(KID)=L(EXIT(x)) x is leavable	L(KID)=L(x)
SEE(x,y)	L(KID)=L(x) y is seeable from x	y recorded as seen
SEARCH(x,y)	L(KID)=L(x)	L(KID)=L(y) or y recorded as seen
SAY(t)	t is text	O(KID)=t [O is output]
HEAR(t)	t is text	I(KID)=t [I is input]
MARK(x,i)	i is KID's (temporary) reference for x	L(x)=L(i) H(x)=H(i)

* * *

The above finishes the specification of the term project given out to the students.

SOME RESULTS

Although "automatic programming" (covering quite a range of objectives) has been studied extensively for many years (see, for example, the lucid review of [3]), no readily available research results would serve as the starting point for the term project described above. It was hoped that the feeling of insecurity generated by such a fact would be counterbalanced by the pride due to doing something original and creative -- at a level of an undergraduate term project!

Each team had to submit a 'progress report' at about mid-semester time. In all fairness, one must say that these reports often contained wishful thinking rather than feasible ideas. For this very reason, they were very useful in guiding the students towards more realistic techniques and goals.

To illustrate the type of results, a segment of the hierarchical definition of BRING (object, destination) is shown in a schematic way, from the working program of one of the teams:

```
PROCEDURE BRING (object, destination)
PICKUP (object, SEARCH (object))
GOTO (destination)
PUTDOWN (object)
```

The same project could at a later stage automatically generate this procedure on the basis of requiring the post-condition

L (object)=destination to be satisfied. The program in fact followed a GPS-like difference-reducing strategy.

Another, rather ambitious, project (almost) taught KID all arithmetic operations, starting out from the axioms of predecessor and successor, and some simple control mechanisms.

One team has implemented the environment for the game of (simplified) soccer and taught KID how to play it.

A linguistically-oriented project made some steps towards teaching KID to analyze sentences, i.e. to denominate subjects, objects, etc. -- when it already knows how to parse sentences, i.e. to denominate nouns, adjectives, adverbs, etc., and is already familiar with the concept of transitivity and with the rule of agreement (between noun and verb, etc.).

Another team worked on, but did not complete, the implementation of a world in which there are several villages, each with many houses and a library. KID can walk to the library of his own village but must use a bike (remember, it was in The Netherlands) to travel to another village. The librarian of each library knows the whereabouts of some books at other libraries, in addition to all the locally available ones. Several copies of a book may exist. The KID gradually learns how to get hold of more and more books through interaction with librarians.

Finally, a successful project interacts with KID in teaching it how to shop at a supermarket. Faulty commands, in conflict with the knowledge already acquired, are not obeyed by KID who asks also for clarification. Also, when the same sequence of instructions is issued the second time, KID reminds TEACH to name the assumed procedure. KID can automatically generalize two or more similar

procedures into one by inserting parameters appropriately.

CONCLUSIONS

It was found a good idea to challenge a class of undergraduate students of good programming background with a term project that required original and creative efforts. Emphasis was to be placed on good design ideas and feasible methods rather than on perfectly running but dull programs.

ACKNOWLEDGEMENTS

The interest and hospitality of Professor R.P. van de Riet, Free University of Amsterdam, and of Professor Th. J. Dekker, the University of Amsterdam, are gratefully adknowledged. I am indebted to Messrs. M. Kessler and G. Sicherman for many discussions about automatically generated programs. Last but not least, the following students (in alphabetical order) have contributed to the subject matter of this paper: D. Biekart, F. Brazier, P. v/d Burgh, H.W.J. Buwalda, C.J. Doedens, N.H.S. Eisma, H.J.C. Gerbscheid, J. Gutter, Y.W. Hendriks, M.P. v/d Hulst, W. Jansweiter, R.M. de Jong, T.L. Kwee, R.M. Langefeld, R. Modderman, B.J. Noordzij, M. v. Someren, J. Swagerman, A.C. Veldkamp, O. Waalemijn, H. Weigand, J.C. Wippler, and E. Zijlstra.

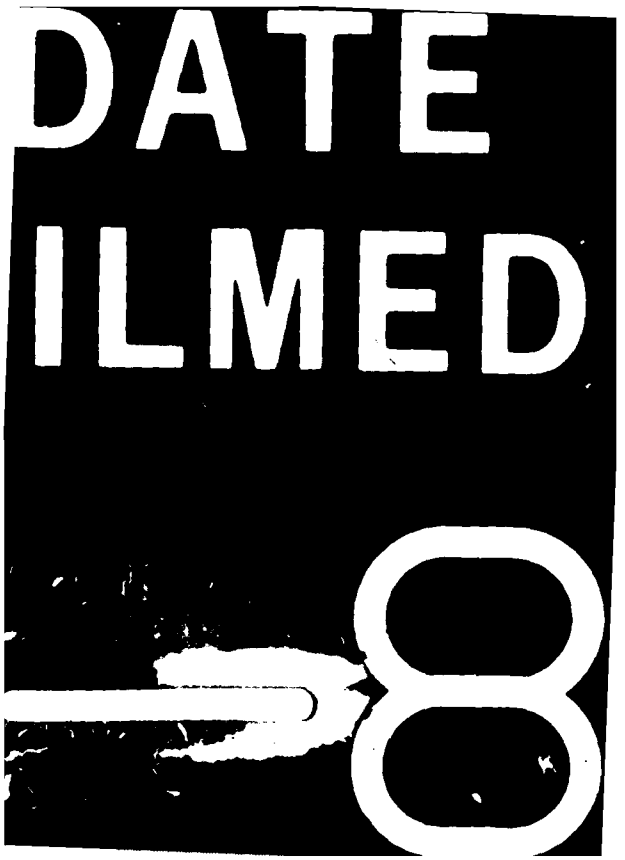
REFERENCES

[1] Findler, N.V.: A few years' experience with courses on non-numerical computation (Proc. IFIP World Conf. on Computer Education, Amsterdam, Holland, pp. II/191-II/194, 1970).

[2] Curriculum '78. Recommendations for the Undergraduate Program

in Computer Science. A Report of the ACM Curriculum Committee on Computer Science (Comm. ACM, 22, pp. 147-166, 1979).

[3] Biermann, A.L.: Approaches to automatic programming. (In M. Yovits and M. Rubinoff (Eds.): Advances in Computers, Vol. 15, pp. 1-63, Academic Press: New York, 1979).


DATE
ILMED
8